Enhancing Adaptive Random Testing in High Dimensional Input Domains

ABSTRACT

Adaptive random testing (ART) is an enhancement of random testing (RT). It can detect failures more effectively than RT when failure-causing inputs are clustered. Having test cases both randomly selected and evenly spread is the key to the success of ART. Recently, it has been found that the dimensionality of the input domain could have an impact on the effectiveness of ART. The effectiveness of some ART methods may deteriorate when the dimension is high. In this paper, we work on one particular ART method, namely Fixed-Sized-Candidate-Set ART (FSCS-ART) and show how it can be enhanced for high dimensional domains. Since the cause of the problems for FSCS-ART may also be valid for some other ART methods, our solutions to the high dimension problems of FSCS-ART may be applicable for improving other ART methods.

1. INTRODUCTION

A major challenge of software development is to establish the correctness of the software. *Software testing* is a major activity of software quality assurance. It assures software quality by actively and systematically detecting faults in order to reduce occurances of software failures.

There exist many testing methods to guide the selection of inputs for test (known as the *test cases*). One simple method is called *Random Testing* (RT), in which test cases are selected in a random manner from the *input domain* (that is, the set of all possible inputs) [11, 18]. There are many merits of using RT in software testing. For example, it can generate numerous test cases automatically at low cost. Its generation of test cases do not need to involve software specifications or source code. It brings "randomness" into the testing process, so it can detect certain failures unable to be revealed by deterministic approaches. Because of these merits, RT has been widely used in industry for detecting failures [7, 8, 10, 15, 16, 17, 19, 20, 21, 22]. In [15, 16], for example, Miller *et al.* used RT techniques to investigate the reliability of standard UNIX utilities. In 1990 [15], they reported that 25% to 30% of these utilities could be crashed. They repeated and extended the study five years later [16], and found that a large number of utility programs were still crashed. RT has also been applied on different software, such as websites [17] and SQL database systems [21]. In [21], a system so-called Random Generation of SQL was developed and it was indicated that the system could "generate valid SQL statements 1 million times faster than a human and execute them".

Despite of the popularity, there are still some critisms against RT. Since it does not make use of any information to generate test cases, it has been regarded by some researchers as ineffective. However, as reported in [1, 2, 9], failure-causing inputs tend to cluster together, and it has been proposed that in such a situation, the *fault-detection effectiveness* of RT can be enhanced by enforcing an even spread of random test cases [13]. This approach was named as Adaptive Random Testing (ART). There are various methods to evenly spread test cases, and all studies conducted [3, 5, 6, 14] so far confirmed that ART has used fewer test cases to reveal the first failure than RT when failure-causing inputs do cluster into contiguous regions (known as the *failure regions* [1]). Since both RT and ART use randomly generated inputs as test cases, it is intuitive to consider ART as an alternative to RT.

Recently, Chen *et al.* [6] have found that the effectiveness of *some* ART methods may deteriorate with the *dimension of the input domain* (that is, the number of input parameters). This problem, known as the high dimension problem, gets worse, when an ART method has a preference for its test cases being generated more frequently in the edges and corners than in the centre of the input domain. This preference is a by-product of some procedures that achieve an even spread of test cases by enforcing them far apart from each other. With such a preference, Chen *et al.* showed that ART could perform even worse than RT when the failure rate is high. Their explanation to such phenomenon is summarized as below. On one hand, when the failure rate is high, the higher the dimension is, the more likely an input in the centre is failure-causing (A detailed justification can be found in [12]). On the other hand, the increase of corners and edges is exponential to the increase of dimensions. Breifly speaking, ART may not be able to select test cases around the centre until most of the corners and edges are filled, and hence ART may take more test cases to detect the first failure than RT.

In this paper, we work on one particular ART method, *Fixed-Sized-Candidate-Set ART* (FSCS-ART) [13] which has a preference of generating test cases in edges and corners, and investigate how FSCS-ART can be enhanced in the high dimensional case. The paper is organized as follows. Section 2 introduces the algorithm of FSCS-ART and the experimental setup related to the study of ART. Section 3 discusses why the performance of FSCS-ART deteriorates with dimensions of the input domain. Section 4 details our approach to enhancing FSCS-ART, and reports our experiment and findings. Section 5 concludes the paper.

2. BACKGROUND

FSCS-ART [13] maintains two sets of test cases, namely, the *executed set*(E) and the *candidate set*(C), where E stores all executed test cases that do not reveal failures, and C stores k random inputs, from which the next test case will be selected. The candidate with the longest distance to its nearest neighbour in E is chosen as the next test case. The algorithm of FSCS-ART is given in Figure 1.

Define n as the number of tests conducted so far.
{
1. Set $n = 0$ and $\mathbf{E} = \{\ \}$.
2. Randomly select a test case, t, from the input domain (according to uniform distribution).
3. Increment n by 1.
4. If t reveals a failure, go to Step 9; otherwise, store t in $\mathbf{E}$.
5. Randomly generate k inputs to construct $\mathbf{C}$ (according to uniform distribution).
6. For each $c_i \in \mathbf{C}$, calculate the distance d_i between c_i and its nearest neighbour in $\mathbf{E}$.
7. Find $c_b \in \mathbf{C}$ such that its $d_b \geq d_i$ where $n \geq i \geq 1$.
8. Let $t = c_b$ and go to Step 3.
9. Return n and t, and EXIT.

}

Figure 1: The algorithm of FSCS-ART

Any faulty program has at least two attributes: *failure rate* (the ratio of the number of failure-causing inputs to the number of all possible inputs) and *failure pattern* (the geometric shapes of the regions formed by the failure-causing inputs and the distribution of these regions within the input domain). Both attributes are fixed upon completion of the coding but unknown to testers before testing.

There are three commonly used metrics to measure the effectiveness of a testing method: E-measure (the expected number of detected failures), P-measure (the probability of detecting at least one failure) and F-measure (the number of test cases required to reveal the first failure). Chen *et al.* [4] have conducted a study on the statistical properties of these metrics for ART. In this paper, we will use F-measure as the effectiveness metrics in order to facilitate the comparison with previous studies. Most of the experimental studies on ART were carried out through simulations. For each simulation study, failure rates and patterns are predefined, and the failure regions are randomly placed in the input domain. A failure is said to be found if a point inside the failure regions is picked. ART is repeatedly applied to such a setting until the simulation has been run a sufficient number of times (S) to achieve a significantly reliable average F-measure (denoted as F_{ART}). The detailed method of obtaining S can be found in [5]. Like all the previous studies, we assume that all inputs have an equal chance of being selected with replacement, and hence, the expected F-measure of RT (denoted as F_{RT}) is theoretically known as $1/\theta$ where θ denotes the failure rate. In this paper, F_{ART} refers to the F-measure of FSCS-ART, unless otherwise specified, and the *ART F-ratio* (that is, F_{ART}/F_{RT})[6] is used as a measurement of the improvement of ART over RT.

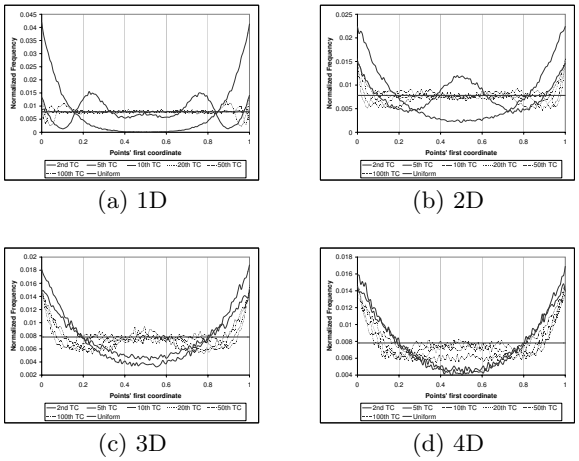
3. HIGH DIMENSION PROBLEM OF FSCS-ART

In this paper, we aim to investigate the high dimension problem of FSCS-ART. We have identified two causes that affect the performance of FSCS-ART. One is related to the preference of test case selection. The other is about the metric (Euclidean distance) used for evenly spreading test cases.

With respect to the first cause, Chen *et al.* [6] have pointed out that FSCS-ART prefers to select test cases from the edges and corners rather than from the centres. We conducted a simple simulation to experimentally verify their hypothesis. If a point is from an edge or corner of the input domain, at least one of its coordinates will be very close to the border of the input domain. Therefore, in this simulation, we only need to check one coordinate of a point to judge its location. Sequences of test cases were generated based on one-, two-, three- and four-dimensional FSCS-ART. The position of a certain coordinate, say the first coordinate, of the nth test case, where $n = \{2, 5, 10, 20, 50, 100\}$ were separately recorded. We performed such runs for 100,000 times and then got six groups of values. Figure 2 shows the frequency distributions for various test cases in one-, two-, three-, and four-dimensional FSCS-ART, respectively. Points generated randomly and according to uniform distribution were also plotted for comparison. The x- and y-axes in the figure denote the first coordinate of a point and the nomalized frequency for the first coordinate value, respectively. Here, the range value of one coordinate is [0,1], so a point is said to reside in an edge or corner if its coordinate is very close to 0 or 1. It is clearly shown that for FSCS-ART, the probability of selecting test cases from the edges and corners are higher than those from the centre, especially for the high-dimensional cases.

The second cause is explained as follows. When D is 1 dimensional, no matter where points (inputs) are located, they will all appear on one line as such, so merely keeping test cases apart in distance is sufficient to achieve an even spread of test cases. However, when D is N dimensional, the spatial distribution of points is more complicated. If FSCS-ART only aims at keeping test cases apart, it cannot fully assure an even spread of test cases all over the input domain. An example is illustrated in Figure 3. Obviously, the test cases in Figure 3(a) are farther apart from one another than those in Figure 3(b). However, it may be controversial to argue that the former is more evenly spread than the latter.

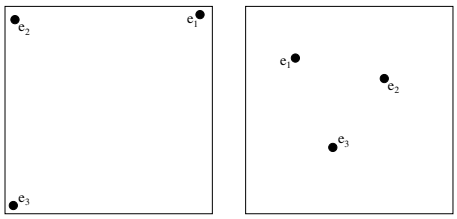
Consider Figure 4 where $C = \{c_1, c_2\}$ (so $k = 2$) and $N = 2$. Assume that c_1 and c_2 have identical distance from their nearest neighbour in E. In other words, they both are entitled to be the next test case according to the existing selection criterion used in FSCS-ART. Further assume there



(a) 1D (b) 2D

(c) 3D (d) 4D

Figure 2: Frequency distribution of the first coordinate of points for FSCS-ART



(a) (b)

Figure 3: Different locations of test cases

exists an element e_i of E such that e_i and c_1 have the same value on y-axis, while no such a relationship exists between c_2 and any element in E. We argue that c_2 should be preferable to c_1 as the next test case. Two reasons are given as below.

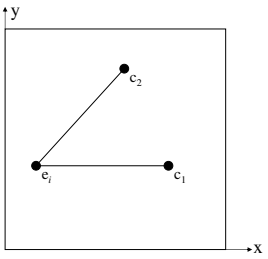


Figure 4: Different candidates with identical distance to e_i

- In addition to keeping test cases apart, intuitively speaking, getting test cases dissimilar in all dimensions should *cover* larger parts of the input space than allowing test cases to be similar in some dimensions. From a *testing coverage* point of view, c_2 should be preferable to c_1.
- Consider the case where the program failures depend on only part of program input parameters (like Program 1). Since failures are only sensitive to Y in Program 1, if we have failed to detect a failure by a test case t, we know that it is better for the follow up test cases different from t with respect to Y. Since it is normally unknown in advance which input parameter (dimension) program failures depend on, to effectively detect failures in all kinds of programs, it is better for the next test case to be different from E (its elements are the inputs unable to reveal failures) as much as possible, not just with respect to the "Euclidean distance" but also with respect to the "value in each input parameter". From this perspective, c_2 should be preferable to c_1.

Program 1 Program failures depend on only one parameter.

```
INPUT X, Y
IF (Y <= 0)
/* ERROR: Should be IF (Y <= 1) */
{  Z = X - Y   }
ELSE
{  Z = X + Y  }
OUTPUT Z
```

In summary, it has been shown that when $N > 1$, using only "distance" as the selection criterion may generate test cases which are neither evenly spread nor effective in detecting failures. In next section, we will present our solutions to address this problem.

4. ENHANCEMENT OF FSCS-ART

For ease of discussion, we introduce the following notations and concepts. In N dimensional D (I_i denotes each of its dimension), the co-ordinates of two points A and B are denoted as (a_1, a_2...a_N) and (b_1, b_2...b_N), respectively. dist(A, B) denotes the Euclidean distance between point A and point B, and $dist_i$(A, B) denotes $|a_i - b_i|$ with respect to I_i. Among all $dist_i$(A, B), the shortest and the longest distance are denoted as minDist(A, B) and maxDist(A, B), respectively. Figure 5 illustrates the above concepts for a 2 dimensional space, with minDist(A, B) being $dist_2$(A, B) and maxDist(A, B) being $dist_1$(A, B). Finally, we define DistRatio(A, B) as the ratio of minDist(A, B) to maxDist(A, B). Obviously, the range value of DistRatio(A, B) is [0, 1]. Intuitively speaking, a larger DistRatio is preferable if other criteria, such as distance, are the same. As an example of illustration, refer to Figure 4 where c_2 is more preferable than c_1. Note that dist(e_i, c_1) = dist(e_i, c_2), but 0 = DistRatio(e_i, c_1) < DistRatio(e_i, c_2) $\leq$ 1.

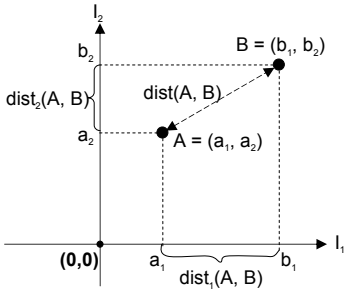


Figure 5: Notations

4.1 Our method

Our enhanced FSCS-ART is basically the same as the original FSCS-ART, but with one additional feature, that is, an eligibility filtering process which is to ensure that the candidates are far apart from previously executed test cases in terms of "input parameters". An input c is eligible if for every e_i of E, DistRatio(c, e_i) is greater than v where v is a value chosen from the range of [0, 1]. For ease of discussion, the condition that determines the eligibility of a candidate is called *eligibility criterion*. In the following paragraphs, we will elaborate our method (namely, *FSCS-ART with filtering by eligibility*) in details. Without loss of generality, we will illustrate this method using a 2 dimensional space.

As an example of illustrating the eligible inputs, consider Figure 6 where e is the only element in E, which is intersected by Lines A, B, C and D having the slope of v, $-v$, $\frac{-1}{v}$ and $\frac{1}{v}$, respectively. In such a scenario, the eligible inputs occupy the dotted regions, and are separated from the ineligible inputs by Lines A, B, C and D.

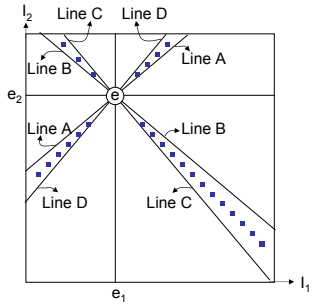


Figure 6: Eligible inputs (dotted regions), v and e (an element of E)

Next, the impact of v and the size of E (|E|) on the number of eligible inputs is investigated. Suppose D consists of 49 elements and |E| = 1 (Figure 7). There are 0, 20 and 36 elements out of 49 elements, which are eligible when v = tan(45°), tan(30°) and tan(15°), respectively. Obviously, the number of eligible inputs increases as v decreases. On the other hand, for a fixed v, the growth of E will "exclude" more and more elements from being eligible. As an example of illustration, refer to Figure 8 where v remains unchanged but the number of elements in E is different (|E| = 1 or 2 in Figure 8(a) or 8(b), respectively). As shown, the number of eligible inputs will decrease with the increase of |E| if v keeps unchanged.

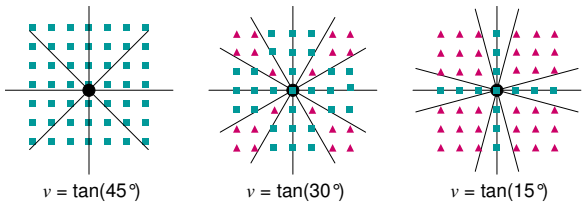
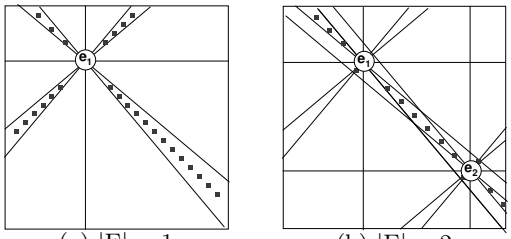


Figure 7: The relationship between v and the number of eligible inputs (represented by triangles)



(a) |E| = 1 (b) |E| = 2

Figure 8: The relationship between |E| and the number of eligible inputs (black spots)

The detailed algorithm of FSCS-ART with filtering by eligibility is given in Figure 9 where Steps 6-14 are introduced to replace Step 5 of Figure 1 (algorithm of FSCS-ART). Basically, we need to construct a candidate set C such that all its elements are eligible. In C, the farthest element away from E is chosen as the next test case.

To use FSCS-ART with filtering by eligibility, the tester needs to set 2 parameters v and r. The role of v has been explained above, and the role of r is explained as follows. Since E grows along with the testing, we will eventually reach a situation where it is impossible or too expensive to construct C. To resolve this problem, we propose to dynamically relax the eligibility criterion during the testing process when insufficient number of eligible candidates has been generated after g attempts. The role of r, which is within the range (0, 1), is to reduce the value of v (by resetting v to be $v \cdot r$) so that the eligibility criterion will be relaxed.

Clearly, the larger the value g is, the longer time it takes to decide whether adjusting v is required or not. In this study, g was arbitrarily set to 4. Since the effect of the filtering disappears when $v = 0$, v is adjusted slowly and only if necessary. Also, we have arbitrarily set that if fewer than 70% candidates are eligible, the current eligibility criterion is regarded to be too strict and hence there is a need to reduce v.

4.2 Experiment 1

It is interesting to know how the effectiveness of FSCS-ART with filtering by eligibility is affected by v and r. We investigate the impact by conducting some simulations. In these simulations, θ was varied from 1 to 0.00005, and the failure region was predefined as a single square which is randomly placed within the input domain.

The results with respect to the setting $v = 0.5(\approx \tan(26.57°))$ and $r = 0.5$ for $N = 2$, 3 and 4 are reported in Figure 10. Under this setting, the experimental data show that the process of filtering does make our method more effective than the original FSCS-ART.

We conducted further experiments with the following settings. In these experiments, D is set to be 4 dimensional.

- v - 0.9 ($\approx \tan(41.99°)$), 0.5 ($\approx \tan(26.57°)$), and 0.1 ($\approx \tan(5.71°)$),

Define n as the number of tests conducted so far.
{
1. Input v and r, where $0 < r < 1$ and $0 \leq v \leq 1$
2. Set $n = 0$, E = { }, C = { }.
3. Randomly select a test case, t, from the input domain (according to uniform distribution).
4. Increment n by 1.
5. If t reveals a failure, go to Step 18; otherwise, store t in E.
6. Randomly generate k inputs to construct **C** (according to uniform distribution).
7. For each $c_i \in$ C, examine the eligibility of c_i. mark c_i "eligible" or "ineligible" accordingly.
8. If all elements of C are eligible, go to Step 15.
9. Set nTrial = 0.
10. Do Steps 11-14 until all c_i of C are eligible
11. Replace each ineligible c_i by another random input.
12. Examine the eligibility of all replacements, and mark them "eligible" or "ineligible" according to the updated value of v.
13. Increment nTrial by 1.
14. After 4 attempts (when nTrial = 4), if fewer than 70% of candidates are eligible, then set nTrial = 0 and $v = v{\cdot}r$.
15. For each $c_i \in$ C, calculate the distance d_i between c_i and its nearest neighbour in E.
16. Find $c_b \in$ C such that its $d \geq d_i$ where $k \geq i \geq 1$.
17. Let t = c_b and go to Step 4.
18. Return n and t. EXIT.
}

Figure 9: The algorithm of FSCS-ART with filtering by eligibility

- r - 0.9, 0.5 and 0.1

There are 9 different scenarios in total. We group the results into Figure 11. Based on these data, we have the following observations

- the higher r is, the smaller the ART F-ratio is
- when r is 0.5 or 0.1, v seems to have little or no impact on the F-measure of FSCS-ART with filtering by eligibility, but when $r = 0.9$, a small v like 0.1 will not deliver an effective FSCS-ART with filtering by eligibility.
- For a given r, any value of v higher than 0.5 will yield similar F-measures.

The first observation is intuitively expected, because a higher r means that v will be decreased more slowly, and hence a more strict elgibility criterion has been imposed. As a consequence, test cases will be enforced more evenly spread over the input domain, and hence the ART F-ratio becomes smaller. The second observation is also understandable because when r is small, no matter what v is used, v will become small very soon (as v will be adjusted by r in Step 14 of the algorithm given in Figure 9). For the last observation, a higher initial value of v yields a small number of eligible inputs. Therefore, v will be reduced quickly until enough eligible inputs are available, so higher values of v will have similar performance. However, it should be noted that generally speaking, a higher v will have smaller F-measure than a lower v. In summary, the above observations imply that effective FSCS-ART with filtering by eligibility needs a high initial value of v (say 0.9) and a large r (say 0.9) in order to keep v large and its approaching to 0 at a slow rate during its application.

4.3 Experiment 2

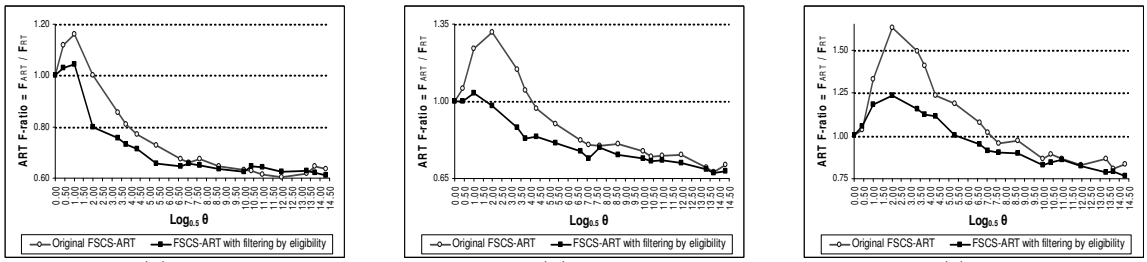
We conducted another experiment to investigate the F-measure of FSCS-ART with filtering by eligibility for the situation where failures depend on some but not all input parameters (this is the situation motivating the development of FSCS-ART with filtering by eligibility). One single rectangular failure region is assumed to reside in a square N dimensional D whose edge length is L. θ is either 0.01, 0.005, 0.001 or 0.0005, and N is either 2, 3 or 4. For a given N dimensional D, the number m of *failure dependent input parameters* (input parameters which cause program failures) is set to be smaller than N and the failure region is assumed to span across $(N - m)$ axes. As an example of illustration, consider a 3 dimensional D and let l_i denote the edge length of the failure region in dimension i (or simply say, l_i is the value range of the failure-causing inputs with respect to the i^{th} parameter). Suppose θ is 0.01. If failures are caused by the 3^{rd} parameter (so $m = 1$), then we have $l_1 : l_2 : l_3 = L : L : 0.01L$. If failures are caused by the 2^{nd} and 3^{rd} parameters (so $m = 2$), then we have $l_1 : l_2 : l_3 = L : 0.1L : 0.1L$.

The simulation results are reported in Tables 1, 2, 3 and 4, which compared FSCS-ART with filtering by eligibility and FSCS-ART with θ being 0.01, 0.005, 0.001 and 0.0005, respectively. In each table, the performances are compared in terms of different combinations of N and m. It is clearly shown that when $m = 1$, no matter how many dimensions the input domain has, the new method has a significant improvement over the original method. When $m > 1$, the new method only marginally outperforms the original method. In summary, our new method also has a better fault-detection effectiveness than the original FSCS-ART when the program failures only depend on part of the input parameters.

5. DISCUSSION AND CONCLUSION

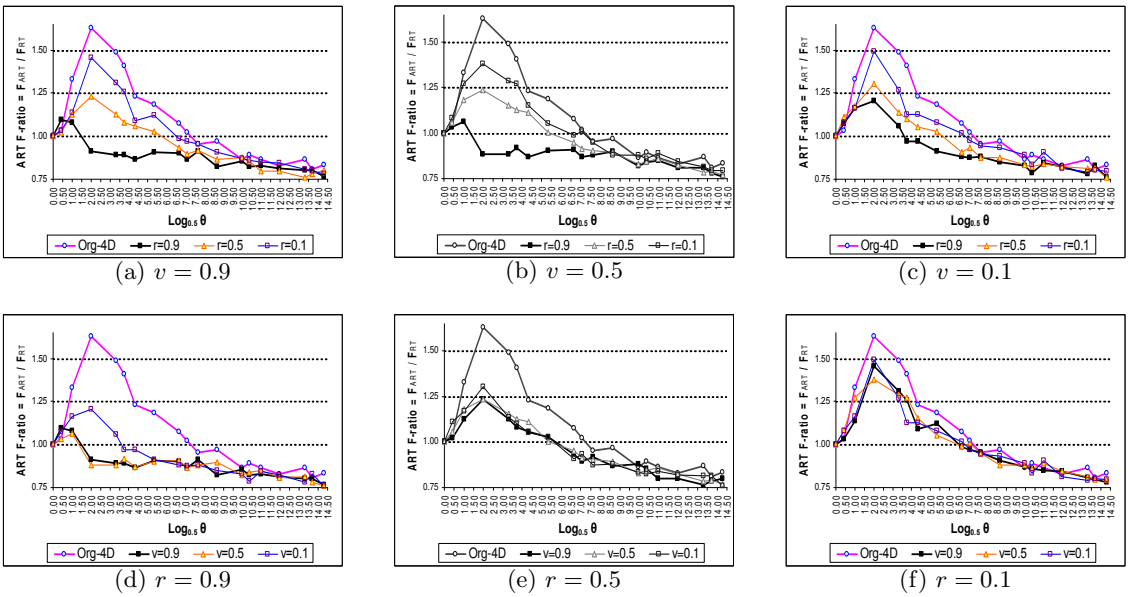
ART was originally proposed to improve the fault-detection effectiveness of RT, especially when the failure-causing input are clustered together. Recently, it was observed that the performance of ART is not so good in higher-dimension situation as that in lower-dimension situation. In this paper, we analysed the high dimension problem of ART, and proposed a new algorithm, FSCS-ART with filtering by eligibility. The results of simulation show that our method improves FSCS-ART not only in high dimensional cases, but also when program failures depend on part of the input parameters. It should be noted that both cases are quite common to occur in real life programs. Therefore, we suggest that FSCS-ART with filtering by eligibility should be used instead of the original FSCS-ART.

In addition, it should be noted that the causes that affect the performance of FSCS-ART may be also applicable to some other ART methods. *Restricted Random Testing* [3] (RRT), for example, also has the preference of selecting test cases from the boundary part of the input domain, as shown



(a) 2D (b) 3D (c) 4D

Figure 10: Comparison between the enhanced version and the original version of FSCS-ART under the settings $v = 0.5$, $r = 0.5$ and $k = 10$



(a) $v = 0.9$ (b) $v = 0.5$ (c) $v = 0.1$

(d) $r = 0.9$ (e) $r = 0.5$ (f) $r = 0.1$

Figure 11: Comparison between the enhanced version and the original version of FSCS-ART

Table 1: Results for the case where program failures depend on only part of input parameters with $\theta = 0.01$

N	m	Method	No. of Run	Ave(F)	StdDev	95% Confidence	
						-	+
2	1	FSCS-ART with filtering by eligibility	1300	69.76	64.11	73.25	66.27
		FSCS	1357	96.08	90.22	100.88	91.28
3	1	FSCS-ART with filtering by eligibility	1274	77.41	70.49	81.28	73.54
		FSCS	1578	103.12	104.43	108.27	97.96
3	2	FSCS-ART with filtering by eligibility	1213	91.59	81.36	96.17	87.01
		FSCS	1171	93.21	81.34	97.87	88.55
4	1	FSCS-ART with filtering by eligibility	1311	82.31	76.01	86.43	78.19
		FSCS	1420	98.37	94.55	103.29	93.45
4	2	FSCS-ART with filtering by eligibility	1468	97.11	94.84	101.96	92.26
		FSCS	1379	108.97	103.22	114.42	103.53
4	3	FSCS-ART with filtering by eligibility	1156	98.49	85.40	103.41	93.57
		FSCS	1129	104.81	89.76	110.05	99.57

Table 2: Results for the case where program failures depend on only part of input parameters with $\theta = 0.005$

N	m	Method	No. of Run	Ave(F)	StdDev	95% Confidence	
						-	+
2	1	FSCS-ART with filtering by eligibility	1203	144.00	127.32	151.20	136.80
		FSCS	1506	189.48	187.53	198.95	180.01
3	1	FSCS-ART with filtering by eligibility	1400	154.63	147.59	162.36	146.90
		FSCS	1357	197.64	185.63	207.52	187.76
3	2	FSCS-ART with filtering by eligibility	1119	175.50	149.76	184.27	166.73
		FSCS	1164	192.79	167.63	202.43	183.15
4	1	FSCS-ART with filtering by eligibility	1386	160.94	152.78	168.99	152.90
		FSCS	1470	197.98	193.61	207.88	188.08
4	2	FSCS-ART with filtering by eligibility	1576	196.27	198.63	206.07	186.46
		FSCS	1490	214.50	211.16	225.22	203.78
4	3	FSCS-ART with filtering by eligibility	1195	195.59	172.35	205.37	185.81
		FSCS	1155	208.93	181.07	219.38	198.48

Table 3: Results for the case where program failures depend on only part of input parameters with $\theta = 0.001$

N	m	Method	No. of Run	Ave(F)	StdDev	95% Confidence	
						-	+
2	1	FSCS-ART with filtering by eligibility	1240	726.93	652.67	763.28	690.58
		FSCS	1488	996.84	980.69	1046.68	947.00
3	1	FSCS-ART with filtering by eligibility	1423	793.34	763.40	833.00	753.67
		FSCS	1543	1020.31	1022.11	1071.33	969.29
3	2	FSCS-ART with filtering by eligibility	1369	954.21	900.58	1001.91	906.50
		FSCS	1425	973.18	937.13	1021.84	924.52
4	1	FSCS-ART with filtering by eligibility	1519	827.06	822.05	868.40	785.72
		FSCS	1476	970.15	950.71	1018.66	921.64
4	2	FSCS-ART with filtering by eligibility	1549	1020.17	1023.71	1071.15	969.19
		FSCS	1417	1044.55	1002.81	1096.76	992.34
4	3	FSCS-ART with filtering by eligibility	1316	970.50	897.62	1019.00	922.00
		FSCS	1315	983.61	909.67	1032.77	934.44

Table 4: Results for the case where program failures depend on only part of input parameters with $\theta = 0.0005$

N	m	Method	No. of Run	Ave(F)	StdDev	95% Confidence	
						-	+
2	1	FSCS-ART with filtering by eligibility	1319	1460.77	1353.13	1533.80	1387.74
		FSCS	1490	1975.79	1944.23	2074.58	1877.00
3	1	FSCS-ART with filtering by eligibility	1342	1605.00	1498.74	1685.19	1524.81
		FSCS	1578	2009.87	2036.24	2110.36	1909.38
3	2	FSCS-ART with filtering by eligibility	1457	1966.32	1913.77	2064.59	1868.05
		FSCS	1491	1987.53	1957.68	2086.90	1888.16
4	1	FSCS-ART with filtering by eligibility	1449	1661.84	1613.08	1744.9	1578.78
		FSCS	1497	2022.04	1994.80	2123.14	1920.94
4	2	FSCS-ART with filtering by eligibility	1548	2096.84	2103.30	2201.62	1992.06
		FSCS	1518	2126.33	2111.64	2232.56	2020.10
4	3	FSCS-ART with filtering by eligibility	1290	1841.60	1686.60	1933.64	1749.56
		FSCS	1213	1927.09	1711.16	2023.39	1830.79

in [12]. And RRT also uses Euclidean distance as the metric of selecting the next test case. Therefore, our solutions to the high dimension problem of FSCS-ART may be applicable for improving other ART methods.

Acknowledgment

This research project is supported in part by an Australian Research Council Discovery Grant (DP0557246).

6. REFERENCES

[1] P. E. Ammann and J. C. Knight. Data diversity: an approach to software fault tolerance. *IEEE Transactions on Computers*, 37(4):418–425, 1988.

[2] P. G. Bishop. The variation of software survival times for different operational input profiles. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS-23)*, pages 98–107. IEEE Computer Society Press, 1993.

[3] K. P. Chan, T. Y. Chen, and D. Towey. Restricted random testing: Adaptive random testing by exclusion. *Accepted to appear in International Journal of Software Engineering and Knowledge Engineering*, 2006.

[4] T. Y. Chen, F.-C. Kuo, and R. Merkel. On the statistical properties of testing effectiveness measures. *The Journal of Systems and Software*, 79(5):591–601, 2006.

[5] T. Y. Chen, F. C. Kuo, R. G. Merkel, and S. P. Ng. Mirror adaptive random testing. *Information and Software Technology*, 46(15):1001–1010, 2004.

[6] T. Y. Chen, F. C. Kuo, and Z. Q. Zhou. On the relationships between the distribution of failure-causing inputs and effectiveness of adaptive random testing. In *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE 2005))*, pages 306–311, Taipei, Taiwan, 2005.

[7] R. Cobb and H. D. Mills. Engineering software under statistical quality control. *IEEE Software*, 7(6):45–54, 1990.

[8] T. Dabóczi, I. Kollár, G. Simon, and T. Megyeri. Automatic testing of graphical user interfaces. In *Proceedings of the 20th IEEE Instrumentation and Measurement Technology Conference 2003 (IMTC '03)*, pages 441–445, Vail, CO, USA, 2003.

[9] G. B. Finelli. Nasa software failure characterization experiments. *Reliability Engineering and System Safety*, 32(1–2):155–169, 1991.

[10] J. E. Forrester and B. P. Miller. An empirical study of the robustness of Windows NT applications using random testing. In *Proceedings of the 4th USENIX Windows Systems Symposium*, pages 59–68, Seattle, 2000.

[11] R. Hamlet. Random testing. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, second edition, 2002.

[12] F.-C. Kuo. On adaptive random testing. PhD thesis, Faculty of Information and Communication Technologies, Swinburne University of Technology, 2006.

[13] I. K. Mak. On the effectiveness of random testing. Master's thesis, Department of Computer Science, University of Melbourne, 1997.

[14] J. Mayer. Lattice-based adaptive random testing. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, pages 333–336, New York, USA, 2005. ACM.

[15] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12):32–44, 1990.

[16] B. P. Miller, D. Koski, C. P. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services. Technical Report CS-TR-1995-1268, University of Wisconsin, 1995.

[17] E. Miller. Website testing. `http://www.soft.com/eValid/Technology/White.Papers/website.testing.html`, Software Research, Inc., 2005.

[18] G. J. Myers. *The Art of Software Testing*. Wiley, New York, second edition, 1979.

[19] N. Nyman. In defense of monkey testing: Random testing can find bugs, even in well engineered software. `http://www.softtest.org/sigs/material/nnyman2.htm`, Microsoft Corporation.

[20] K. Sen, D. Marinov, and G. Agha. Cute: a concolic unit testing engine for C. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 263–272, New York, NY, USA, 2005. ACM Press.

[21] D. Slutz. Massive stochastic testing of SQL. In *Proceedings of the 24th International Conference on Very Large Databases (VLDB 98)*, pages 618–622, 1998.

[22] T. Yoshikawa, K. Shimura, and T. Ozawa. Random program generator for Java JIT compiler test system. In *Proceedings of the 3rd International Conference on Quality Software (QSIC 2003)*, pages 20–24. IEEE Computer Society Press, 2003.